

Proposal: Kylin storage on Parquet (KYLIN-3621)

Shaofeng Shi 2018/10/16

Agenda

- Background
- Why Parquet
- Overview
- Encoding in Parquet
- Parquet file schema
- File layout
- Query engine for Parquet
- Other enhancements

Background

- So far HBase is the only storage engine for Kylin;
- HBase is not a columnar storage and has no secondary index;
- HBase does not fit for cloud deployment and auto-scaling;
- Goal: find a light-weighted, columnar, and cloud-friendly storage for Kylin.

Why Apache Parquet

- Open source columnar file format
- Efficient I/O, high compression
- Well integration with Hadoop, Hive, Spark, Impala and others
- Work with most FS including HDFS, S3, Blob store, OSS
- Mature and stable

Kylin with Parquet: Overview

- Parquet files in HDFS/S3, getting replication, consistency, durability and other benefits;
- Use Spark (or other computing framework) as the query engine to process parquet in parallel;
- Pre-filtering and skipping with the file, block, page level index;
- Only read the necessary columns from disk;
- Pushdown aggregation to Spark;
- Final processing in Kylin with Calcite.

Cube encoding in Parquet

- Dimensions
 - For dictionary encoded dimension, save encoded integer in Parquet
 - For fixed-length encoding, save origin string in Parquet
- Measure
 - For simple numeric measures (SUM, MIN, MAX, etc), save data in original data type in Parquet
 - For complex measures (COUNT DISTINCT, TopN, Percentile, etc), save binaries in Parquet

Parquet file schema

- Different cuboids have different columns (schema);
- Compact mode vs Sparse mode;
- Adding cuboid ID to file/folder name, or parquet metadata, or as additional column
- Parquet schema sample:
 - |Dim1|Dim2| ... |DimN|Measure1|Measure2|...|MeasureN|CuboidID|

Parquet schema: Compact mode

- Each Cuboid has its own schema;
- Only the column in the cuboid will appear in the parquet;
- Advantage
 - Compact in size
- Disadvantage
 - Code is a little complex
 - Different cuboids couldn't be processed together

Parquet schema: Sparse mode

- All columns appear in the parquet;
- Absent columns will all be null;
- Advantage
 - Simple
 - Can process multiple cuboids in one batch;
- Disadvantage
 - A little metadata overhead

File layout

- Storage layout is important for I/O optimizations
- Do as much as possible pruning before read the file
 - Filter by folder, file name, etc
- Candidate plans
 - Plan A: folder by cuboid ID
 - Plan B: all in one folder, cuboid ID as file name prefix
 - Plan C: folder by cuboid layer, cuboid ID as file name prefix
 - Plan D: cuboids stored in buckets, all in one folder, with additional mapping

Plan A

- Each Cuboid uses a dedicated folder
- Cube
 - Segment A
 - Cuboid-1111
 - abc.parquet
 - def.parquet
 - ...
 - Cuboid-1001
 - abc.parquet
 - ...
 - Segment B
 - Cuboid-1111
 - abc.parquet
 - ...

Plan A

- Advantage
 - Filter by folder is good enough;
 - Can dynamically add/remove cuboid without impact others
- Disadvantage
 - Many folders when the cube has many cuboids.
 - Small cuboids cause the same number small files;

Plan B

- All cuboids store in the same folder;
- Put Cuboid ID as file name prefix; Filter by file name prefix;
- Cube
 - Segment A
 - 1111-abc.parquet
 - 1110-abc.parquet
 - 1101-abc.parquet
 - ...
 - Segment B
 - ...

Plan B

- Advantage
 - Fewest folders
- Disadvantages
 - Not good for by-layer building, or on-demand adding/removing cuboids, need to copy or move files;
 - The folder may have a long file list, causing the list and filter by file name is slower.
 - Small cuboids cause the same number small files;

Plan C

- By layer: all cuboids of the same layer in the same folder
- Filter by folder and then by file name prefix;
- Cube
 - Segment A
 - layer-4
 - 1111-abc.parquet
 - layer-3
 - 1110-abc.parquet
 - 1101-abc.parquet
 - ...

Plan C

- Advantage
 - Less folder numbers than Plan A;
 - Good for by-layer processing;
 - List and filtering is better than Plan B;
- Disadvantage
 - Small cuboids cause the same number small files;
 - Not good for on-demand adding/removing cuboids, need to copy or move files to existing folder;

Plan D

- Parquet files as buckets; bucket # is determined by total size;
- Put cuboid to 1 or multiple buckets depends on its size;
- Record the cuboid to bucket mapping in metadata;
- Cuboid ID as a column in Parquet schema, and need be added as a query condition;
- Cube
 - Segment A
 - Bucket1.parquet
 - Bucket2.parquet
 - ...

Plan D

- Advantage
 - Fewest folders and files;
 - Files are close in size; No small files;
- Disadvantage
 - Has additional I/O; multiple cuboids may in the same parquet page;
 - Metadata and building job are a little complicated compared to others;
 - Not good for on-demand adding/removing cuboids.

Storage layout: a comparison

	Less folder or small file	On-demand building	Job friendly	I/O efficiency	Cloud friendly
Plan A		Excellent	Good	Good	Good
Plan B	Good			Good	
Plan C	Good		Excellent	Good	Good
Plan D	Excellent				

Folder/file number: less folders and small files

File size: less file number, and closed file size

Job friendly: good for batch processing

I/O efficiency: less read

Cloud friendly: can direct-output, no file move;

Plan A, C seems be more balanced.

Query engine for Parquet: Overview

- Start a long running Spark application on Kylin startup, as the backend service
- Read the cuboid parquet file as a Dataset;
- Convert Kylin GTScanRequest to Spark Dataset operations (where, group, etc);
- Use Kylin UDAF for complex measure aggregations;
- Decode cuboid data and then return to Calcite.

Convert Calcite execution to Spark jobs

- Make post calculation also distributed;
- Break bottleneck, especially for complex queries (union, join, etc)
- This is a nice-to-have feature, not required at first phase.

Other enhancements

Configurable path pattern

- Kylin should allow customizing the path logic, for fitting different file storage;
- For HDFS, this pattern is good:
 - /kylin/metadata_name/cube_name/segment_name/file1.parquet
- For S3, should use other patterns like:
 - /<salt>/kylin/metadata_name/cube_name/segment_name/file1.parquet
 - or /segment_name/cube_name/metadata_name/kylin/file1.parquet

Pluggable external index

- Kylin should allow pluggable index as a supplement to Parquet index;
- External index can be built when cube is generated, or later based on query patterns;
- Absence of external index won't break query execution.

Question?

- Welcome to join the discussion, please send your comment or question to dev@kylin.apache.org